(19) Europäisch s Patentamt
European Patent Office
Offic uropé n d s br vets

(11) EP 0 570 137 B1

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
22.07.1998 Bulletin 1998/30

(51) Int Cl.[6]: **G06F 3/023**, G06F 3/033

(21) Application number: **93303392.0**

(22) Date of filing: **29.04.1993**

(54) Icon driven computer apparatus

Ikongesteuerter Computer

Ordinateur commandé à l'aide d'icônes

(84) Designated Contracting States:
DE FR GB

(30) Priority: **11.05.1992 US 880822**

(43) Date of publication of application:
18.11.1993 Bulletin 1993/46

(73) Proprietor: **International Business Machines Corporation**
Armonk, N.Y. 10504 (US)

(72) Inventors:
- **Shrader, Theodore Jack London**
  Austin, Texas 78728 (US)
- **Scully, Keith James**
  Austin, Texas 78758 (US)

(74) Representative: **Burt, Roger James, Dr.**
IBM United Kingdom Limited
Intellectual Property Department
Hursley Park
Winchester Hampshire SO21 2JN (GB)

(56) References cited:
EP-A- 0 473 524
WO-A-89/11695
US-A- 5 005 119
US-A- 5 021 976

- **RESEARCH DISCLOSURE no. 332, December 1991, EMSWORTH, GB page 911 ANONYMOUS 'Abstract no. 33210, Duplicator icon for workstation applications'**

EP 0 570 137 B1

Printed by Jouve, 75001 PARIS (FR)

Description

The invention as it is defined in the appended claims relates to computer apparatus.

Using icons to organize a computer display and giving a user the appearance of an lectronic desktop first appeared at Xerox Parc in the late 70's. Early attempts to commercialize a desktop motif were unsuccessful. However, an implementation is disclosed in US-A-5060135, "Apparatus for Manipulating Documents in a Data Processing System Utilizing Reduced Images of Sheets of Information Which Are Movable." More recently, the IBM OS/2 Workplace Shell desktop motif provides icon manipulation. For example, users can drag a document icon, representing a letter, and drop it on the shredder icon to delete the document.

Other examples of icon manipulation include, US-A-5021976 entitled, "Method and System for Generating Dynamic Interactive Visual Representations of Information Structures Within a Computer," which discloses a knowledge based system that links various mathematical relationships with icons on a graphical display. Visual features of the display assume various conditions (change of colour/size) in accordance with the mathematical relationships.

In EP-A-473524 a 'rubber stamp' metaphor is employed. Icons representing various types of rubber stamp are used in order to allow a user to imprint a property or value on a data object.

However, none of the prior art techniques provide a method for easily and flexibly manipulating the information in an icon from one form to another in accordance with the subject invention.

This invention enables computer apparatus comprising a display device, information storage means and a processor responsive to a predetermined user input operation to process stored information according to the location of associated icons on the screen of the display device, characterised in that at least one of said icons is a transformer icon arranged to represent a transformation operation which can be performed on stored information associated with one or more input icons, and to enable the transformation operation to be invoked by a predetermined user input operation when the, or each, input icon is positioned over the transformer icon, the apparatus comprising means to generate one or more output icons, distinct from the transformer icon and associated with the transformed information.

Thus a method is provided in which icon transformers can be used in an object-oriented, graphical environment to change the contents of an icon from one state to another using drag and drop techniques.

In one embodiment, this is accomplished by creating a transformer object, represented by an icon, that has an associated set of rules stored in an associated data structure. This transformer object modifies the attributes of a dropped icon and changes its contents to a new object, while retaining some of the attributes from the original object, if needed. If a user drops an icon object on a transformer icon and the dropped object does not conform to the input expected by the transformer, the input icon is transformed into a not symbol.

For valid input object processing, when the user drops an object on a transformer object, it is removed from the desktop, application window, or container object from which it came. In its place, an output object icon is created next to the icon transformer object. This output object icon's contents are a modified version of the dropped input object's contents. The contents are changed in accordance with the rules in the transformer object.

Users can also duplicate their input objects and drop them separately or in a group on top of an icon transformer. The transformer changes one or more of the input objects into their output object counterparts based on the number of objects dropped on the transformer.

Objects can be changed from one state to another in current application implementations. For example, when importing a text file from word processor X to word processor Y, the latter word processor converts a document from X into one that Y can understand. This technique involves changing the contents of the file and/or its attributes. However, this processing does not address the problem in an object-oriented manner. These actions take place within the application and not outside of it, such as on the desktop or in a window container. Icon transformers can exist where icons can be dragged and dropped to convert objects in a one-to-one, many-to-one, one-to-many, or many-to-many fashion.

An embodiment of the invention will now be described, by way of example, with reference to the accompanying drawings, wherein:

Figure 1 is a block diagram of a personal computer system in accordance with the subject invention;

Figure 2 illustrates a file input icon and a converted output icon before and after a conversion operation in accordance with the subject invention;

Figure 3 is a data structure representing an icon in accordance with the subject invention;

Figure 4 is a data structure representing a transformation icon in accordance with the subject invention;

Figure 5 is a flowchart of the icon transformation logic in accordance with the subject invention;

Figure 6 is a flowchart of the icon transformation logic in accordance with the subject invention; and

Figure 7 is a flowchart of the transformation logic in accordance with the subject invention.

The invention is preferably practised in a representative hardware environment as depicted in Figure 1, which illustrates a typical hardware configuration of a computer workstation having a central processing unit 10, such as a conventional microprocessor, and a number of other units interconnected via a system bus 12. The workstation shown in Figure 1 includes a Random Access Memory (RAM) 14, Read Only Memory (ROM) 16, an I/O adapter 18 for connecting peripheral devices such as disk units 20 and tape drives 40 to the bus, a user interface adapter 22 for connecting a keyboard 24, a mouse 26, a speaker 28, a microphone 32, and/or other user interface devices such as a touch screen device (not shown) to the bus, a communication adapter 34 for connecting the workstation to a data processing network and a display adapter 36 for connecting the bus to a display device 38.

A method and system is provided for easily changing information associated with an icon from one state to another. For example, a user may require that a document be changed from one word processing format to another (Revisable Form Text (RFT) to WordPerfect). Today, this would be accomplished by invoking a CONVERT program from the command line or from an internal application call and inputing the source and destination file for the conversion. Then, the program would process the source information and convert the file into the destination output file.

The subject invention allows a user to convert a file by simply dragging the file icon to an icon representative of a convert operation and dropping the file icon onto the convert icon. This action will invoke a convert operation that transforms the original file associated with the file icon into a converted file associated with a new icon representative of the converted file. Figure 2 shows the file icon 200 and the convert icon 210 before the conversion operation, and the convert icon 220 and the converted file icon 230 after the convert operation. Note that the shape of the icon has changed.

Figure 3 is a data structure representing the icon. Attribute information 300 contains descriptive information pertaining to the icon similar to a header in a data file. The contents 310 are similar to text in a data file.

Figure 4 is a data structure representing a transformation icon. Attribute information 400 contains descriptive information pertaining to the icon similar to a header in a data file. The contents 410 contain rules associated with a transformation operation that is invoked when an icon is dropped on the transformation icon.

Figure 5 is a flowchart depicting detailed logic in accordance with the subject invention. Processing commences at terminal block 500 and immediately passes to function block 510 where the icon transformer retrieves attributes of an icon dropped on it. Then a test is performed at decision block 512 to determine if the dropped icon has compatible attributes. If compatibility is detected, then the dropped icon is removed from the desktop at function block 514, the transformation code is executed at function block 516 and control passes via label 520 to label 570 where the transformed icon is redisplayed next to the icon transformer as shown in function block 580. The desktop metaphor in this example can be an application window or a container object similar to that found in the OS/2 operating system.

If the attributes do not match at decision block 512, then further tests are performed in a similar fashion leading up to decision block 530 to determine if the attributes of the dropped icon match another characteristic that the icon transformer can handle. If so, then the dropped icon is removed from the display at function block 532, transformation code associated with the characteristic is executed at function block 534 and the transformed icon is redisplayed next to the icon transformer at function block 580. If no match is detected between the dropped icon's attributes and any of the transformer characteristics, then the dropped icon is redisplayed in an error mode (such as turning it into a not symbol) as shown in function block 550.

Figure 6 is a flowchart of the detailed logic in accordance with the subject invention. Processing commences at terminal 600 and control immediately passes to function block 602 where attributes of the icon dropped on the icon transformer are retrieved. Then, at decision block 604 a test is performed to determine if the attributes of the dropped icon match the expected attributes for a document created by a word Processor X. If a match is detected, then the document icon is removed from the desktop as shown in function block 606, the special characters that were imbedded by X are replaced with spaces as set forth in function block 608 and control is passed via label 610 to Figure 7 at label 720. In Figure 7 at label 730, the transformed document icon is redisplayed and the information associated with the transformed icon is now an undelimited ASCII file.

If the attributes of the dropped icon do not match at decision block 604, then another test is performed at decision block 612 to determine if the attributes of the dropped icon match the expected attributes for a document created on a text processor Y that uses global variables. If so, then the document icon is removed from the desktop at 614, the special characters imbedded by Y are replaced with blank spaces at function block 620 and control is passed via label 622 to label 700 of Figure 7. In Figure 7, processing resumes at function block 710 where each global variable is replaced by the value assigned by a user and the transformed icon is redisplayed with its modified underlying contents at function block 730.

If no match is detected at decision block 612, then in decision block 630 a test is performed to determine if the

attribute of the dropped icon match with a delimited ASCII file. If so, then the document icon is removed at function block 632, the special characters are removed at function block 634, control is passed via label 640 to label 720 and processing is completed at function block 730 as described above. If no match is detected, then the icon is modified to present an error to the user at function block 642 and control is returned at terminal 650.

The following code was written in Smalltalk/V PM for the OS/2 environment. TransExample (the Transformation Example) is the main class where the open method can be run. Its supporting classes are DragTest class where rendering and target and source transferring methods are stored and the TransItem class which is the parent class for the drag/drop objects involved in the prototype.

The TranslFile subclass represents the input file to the transformer (or the object that will be dragged and dropped on the transformer object). The TransOFile subclass represents the output file from the transformer (or the object that is created from an object that was dragged and dropped on the transformer object). The Transformer subclass represents the object icon that transforms the input object to the output object. It also contains the rules to change the contents of dropped objects.

The drag objects were created to represent the input file object (the one to be transformed), the transformer object (the one that performs an action on the contents of the dropped object), and the output file object (the object after it has been transformed). Note that the output file object retains many of the attributes of the input file object, except most notably, the contents of the object changes as well as the shape, size, and or hues of the icon. This visually signifies that the object underwent a transformation. Note that transformations need not take place only on document or text objects. They can also occur on other objects as well, so long as there is a defined process of transformation from the input to the output object.

TRANSFORMATION CLASS SOURCE CODE

```
ViewManager subclass: #TransExample
instanceVariableNames:
    'container '
  classVariableNames: ''
  poolDictionaries:
    'PMConstants PMContainerConstants ' !

!TransExample class methods ! !

!TransExample methods !
 defineFieldsIn: aContainer
                "Define the detail view fields
for the container."
        | fields |
        fields := OrderedCollection new.

        fields
                add: (ContainerDetailField new
                                name: #pszIcon);
                add: (ContainerDetailField new
                                name: #hptrIcon;
                                containsGraphic ).
        aContainer fields: fields asArray!

delete: aContainer
                "Delete the selected item in aContainer."
        | items name |
        items := aContainer selectedItems.
        items do: [:item |
                item class == TransIFile ifTrue: [
                        name := item
```

```
containerName, item name.
                                ( MessageBox
confirm: 'Delete ', name, '?' ) ifTrue: [
aContainer deleteItem: item ]]].!
drop: item into: aContainer at: anOffset
                "Drop an item into aContainer as appropriate
                for text or icon display mode at anOffset point."
        | matching newItem |
        matching := (aContainer itemsMatching: item name)
                select: [:each| each name
asUpperCase = item name asUpperCase].
        ( ( aContainer dragData at: 4 ) = CnDragover )
                ifTrue: [matching do: [:each|
aContainer deleteItem: each].
                ((aContainer dragData at: 2 )
notNil and:
                        [(( aContainer dragData
at: 2 ) name) = 'TransFile'])
                        ifTrue:[newItem :=
TransItem className: 'OutFile' transformedName: item name.
                                newItem
                                allocateFieldsIn:
aContainer;
                                position: ( ( (
aContainer dragData at: 3 )
                                dropCoord
mapScreenToClient: aContainer )
                                        +
anOffset  + aContainer workSpaceOrigin ).
                                aContainer
insertItem: newItem.
                                ]
                        ifFalse: [newItem :=
TransItem className: item name ftype: item ftype.
                                newItem
                                allocateFieldsIn:
aContainer;
```

```
                                    position: ( ( (
aContainer dragData at: 3 )
                                    dropCoord
mapScreenToClient: aContainer )
                                                +
anOffset  + aContainer workSpaceOrigin ).
                                    aContainer
insertItem: newItem.
                                    ]
                            ]
                            ifFalse: [matching do:
[:each| aContainer deleteItem: each].
                                    newItem :=
TransItem className: item name ftype: item ftype.
                                    newItem
allocateFieldsIn: aContainer;
                                                position:
anOffset + ( aContainer extent // 2 ).

                                    aContainer
insertItem: newItem after:
                                            ( (
aContainer dragData at: 2 ) isNil
ifTrue: [ CmaFirst ]
ifFalse: [ aContainer dragData at: 2 ] ).
                            ]!

droppingItems: aContainer
                "Items are scheduled to be dropped into aContainer."
        | target |
        target := aContainer dragData at: 2.
        ( target notNil and: [ ( aContainer
dragData at: 4 ) = CnDragover ] )
                ifTrue: [aContainer dragDrop
target: ( target name ) ].!

insertItemsIn: aContainer
                "Insert all of the directory items in of the current
```

```
directory
                into aContain r."
        | items |
        items := OrderedCollection new.
        items add:
                (TransIFile new
                        name: 'InFile1';
                        ftype: 'I').
        items add:
                (TransIFile new
                        name: 'InFile2';
                        ftype: 'I').
        items add:
                (TransIFile new
                        name: 'InFile3';
                        ftype: 'I').
        items add:
                (Transformer new

                        name: 'TransFile';
                ftype: 'T').

        aContainer contents: items.!

menu: aContainer
                "Add the view changing menu to the menu bar."
        aContainer setMenu: ( Menu new
                appendSubMenu: ( Menu new
                        appendItem: 'Normal'
selector: CvIcon ;
                        appendItem: 'Name'
selector: CvName ;
                        appendItem: 'Name/Flow'
selector: CvName | CvFlow ;
                        appendItem: 'Text'
selector: CvText ;
                        appendItem: 'Text/Flow'
selector: CvText | CvFlow ;
```

```
                        appendItem: 'Tree Icon'
selector: CvTree | CvIcon ;
                        appendItem: 'Tree Name'
selector: CvTree | CvName ;
                        appendItem: 'Tree Text'
selector: CvTree | CvText ;
                        owner: container;
                        title: 'Icon';
                        yourself );
                appendItem: 'Detail' selector:
CvDetail | CaDetailsviewtitles;
                owner: container;
                title: 'View';
                selector: #view:;
                yourself )!

open
                "Open a TransExample on the directory named aPath

                TransExample new open

        "
        | dragger cp |
        self label: 'TransExample'.
        self mainView style: ( self mainView
defaultFrameStyle | FcfShellposition ).
        self addSubpane: (cp := ContainerParent new).
        cp addSubpane: ((container := Container new)
                owner:self;
                title: 'Icon Transformation
Example';
                when: #getFields perform:
#defineFieldsIn: ;
                when: #getContents perform:
#insertItemsIn: ;
                when: #getMenu perform: #menu: ;
                when: #startDrag perform:
#startDrag: ;
```

```
                when: #dr pRequest perform:
#droppingItems: ;
                when: #dragComplete perform:
#wasDragged: ;
                when: #dropComplete perform:
#wasDropped: ;
                when: #delete perform: #delete: ;
                when: #select perform: #selected:
).

        container setAttributes:
CaMixedtargetemph.

        dragger := DragDrop for: container.
        dragger
                container: 'TransExample';
                mechanisms: ( Array with:
DragTest new ).
        container dragDrop: dragger.
        self openWindow!

selected: aContainer

                "The user has selected an item in
aContainer.  If a
                tree view is displayed, insert
the children of the selected
                item into the container."
        | item cnrInfo |
        cnrInfo := container queryCnrInfo.
        ( cnrInfo flWindowAttr bitAnd: CvTree ) = 0
                ifFalse: [item := aContainer selection.
                        ( aContainer isSelected:
item )
                ifTrue: [ item
insertChildrenInto: aContainer ]]!

startDrag: aC ntainer
```

```
                "Start a drag and drop sessi n."
        | dragItems |
                ( dragItems := aContainer
dragList ) isEmpty
                        ifFalse: [ aContainer
dragDrop drag: dragItems ]!

textContents: aPane
        aPane contents: 'this is some text'!

view: aCvConstant
        container view: aCvConstant!

wasDragged: aContainer
                "An item was dragged inside
aContainer."
        ^nil!

wasDropped: aContainer
                "An item was dropped inside
aContainer."
        | item offset |
        offset := 0@0.

        aContainer dragDrop items do: [ :dragItem
|
                item := TransItem className:
dragItem name ftype: (dragItem userInfo) ftype.
                item notNil ifTrue: [
                                self drop: item
into: aContainer at: offset.
                                        offset :=
offset + 12 ].
        ]! !
```

DRAG TEST CLASS SOURCE CODE

```
DragTransfer subclass: #DragTest
```

```
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries:
    'PMDragConstants PMConstants ' !

!DragTest class methods ! !

!DragTest methods !
renderingMechanism
                "Answer a string which describes
the rendering
                mechanism implemented."
        ^'DRM_OS2FILE'!

sourceTransfer: item
                "Private - The source will render the item."
        | dragTransfer |
        dragTransfer := PMDragTransfer size: 1.
        dragTransfer
                cb: PMDragTransfer sizeInBytes;
                hwndClient: owner owner handle;

                pditem: item pmItem contents
contents;
                selectedRMF: ( '<', self
renderingMechanism, ',', item format first, '>'
);
                renderToName: ( owner target,
item pmItem targetName );
                ulTargetInfo: 0;
                usOperation: owner operation;
                fsReply: 0.

        ( dragTransfer sendMsg: DmRender to: item
pmItem hwndItem with: 0 )
                ifTrue: [ ^nil ]
                ifFalse: [item pmItem
                        sendTransferMsg:
```

```
DmEndconversation
                        response: DmflTargetfail.
                        owner freeTransferItem:
item
                ]!

targetTransfer: item
                "Private - The target will
perform the rendering
                operation without the direct
involvement of the source."
        ( self transfer: item to: ( owner target,
item name ) )
                ifTrue: [item pmItem
                        sendTransferMsg:
DmEndconversation
                        response:
DmflTargetsuccessful.
                ]
                ifFalse: [item pmItem
                        sendTransferMsg:
DmEndconversation
                        response:
DmflTargetfail.
                ].

        owner freeTransferItem: item!

transfer: item
                "Private - Begin the direct
manipulation operation."
        | hstrSrc tmpDir |
        owner target: owner target.
        hstrSrc := PMHandle fromBytes: item
pmItem hstrSourceName.
        ( hstrSrc = NullHandle or: [ ( owner
isNativeFormat: item ) not ] )
                ifTrue: [ self sourceTransfer:
```

```
item ]
                ifFalse: [ self targetTransfer:
item ]!

transfer: item to: dest
                "Private - Perform the actual
operation and return true if successful."
        | src operation |
        operation := owner operation.
        src := item pmItem containerName, item
pmItem sourceName.
        ( src asUpperCase = dest asUpperCase )
        ifTrue: [ ^true ].
    ^true! !

                TRANSFORMATION ITEM SUBCLASS
SOURCE CODE


ContainerItem subclass: #TransItem
  instanceVariableNames:
    'ftype name '
  classVariableNames: ''
  poolDictionaries:
    'PMConstants PMContainerConstants ' !


 !TransItem class methods !
className: aName ftype: aType
                "Answer a TransIFile, TransOFile
or Transformer
                which corresponds to the ftype
aType."
         (aType = 'O')
                ifTrue: [
                        ^TransOFile new
                                name: aName;
                                ftype: 'O';
```

```
"Output/Transformed File"
                                        yourself ].

        (aType = 'T')
                ifTrue: [
                        ^Transformer new
                        name: aName;
                        ftype: 'T';
"Transformation Icon"
                        yourself ]
                ifFalse: [
                        ^TransIFile new
                        name: aName;
                        ftype: 'I';  "Input file"
                        yourself ].!

className: aName transformedName: aTName
                "Answer a TransOFile which
corresponds to the
                path aString."
        (aName = 'OutFile')
                ifTrue: [
                        ^TransOFile new
                        name: aTName;
                        ftype: 'O';
"Output/Transformed File"
                        yourself ].! !

!TransItem methods !
containerName
                "Answer the name of the container
this item is contained in.  The name is the
                name used by the drag and drop
protocols."
        ^''TransExample''!

deleteChildTreesIn: aContainer
                "Private - Delete all subtrees
```

```
rooted by the receiver in
                aContainer."
        ^nil!

ftype
                "Answer the ftype item of the
receiver item."
        ^ftype!

ftype: aTransItem
                "Set the ftype of the receiver to
aTransItem."
        ftype := aTransItem!

insertChildrenInto: aContainer
                "Private - insert all of the
immediate children of the receiver
                into aContainer."
        ^nil!

name
                "Answer the name string of the
receiver."
        ^name!

name: aString
                "Set the name string of the receiver."
        name := aString.
        super name: aString! !

                TRANSFORMATION INPUT FILE ITEM
SUBCLASS SOURCE CODE

TransItem subclass: #TransIFile
  instanceVariableNames: ''
  classVariableNames:
    'Icon '
  poolDictionaries: '' !
```

```
!TransIFile class methods !

aboutToSaveImage
        Icon := nil! !

!TransIFile methods !

icon
                "Return icon to be transformed"
        Icon := CursorManager pointerFromModule:
'dricon' id: 2.
        ^Icon! !

           TRANSFORMATION TRANSFORMER SUBCLASS
SOURCE CODE

TransItem subclass: #Transformer
  instanceVariableNames: ''
  classVariableNames:
    'Icon '
  poolDictionaries:
    'PMContainerConstants ' !

!Transformer class methods !

aboutToSaveImage
        Icon := nil! !

!Transformer methods !

attributes
                "Answer the item attributes.  In
this case allow dropping onto
                the receiver."
        ^CraDroponable!

icon
```

```
                "Return icon of the icon
transformer."
        Icon := CursorManager pointerFromModule:
'dricon' id: 3.
        ^Icon!

insertChildrenInto: aContainer
                "Private - insert all of the
immediate children of the receiver
                into aContainer."
        ^nil!

isChildrenCurrent
                "Private - Answer true if the
children of the receiver matches the current
                folder contents in the receiver,
otherwise false."
        ^nil!

isDroponable
                "Answer true if the receiver can
have items dropped on it, otherwise false."
        ^true! !

TRANSFORMATION OUTPUT FILE SUBCLASS SOURCE CODE

TransItem subclass: #TransOFile
  instanceVariableNames: ''
  classVariableNames:
    'Icon '
  poolDictionaries: '' !

!TransOFile class methods !

aboutToSaveImage
        Icon := nil! !

!TransOFile methods !
```

```
icon
                    "Return transformed icon"
        Icon := CursorManager pointerFromModule:
'dricon' id: 60.
        ^Icon! !
```

While the invention has been described in terms of a preferred embodiment in a specific system environment, those skilled in the art recognize that the invention can be practiced, with modification, in other and different hardware and software environments within the scope of the appended claims.

Claims

1. Computer apparatus comprising a display device (38), information storage means (14, 16, 20) and a processor (10) responsive to a predetermined user input operation to process stored information according to the location of associated icons on the screen of the display device (38), characterised in that at least one of said icons is a transformer icon arranged to represent a transformation operation which can be performed on stored information associated with one or more input icons (200), and to enable the transformation operation to be invoked by a predetermined user input operation when the, or each, input icon is positioned over the transformer icon (210), the apparatus comprising means to generate one or more output icons (230), distinct from the transformer icon and associated with the transformed information.

2. An apparatus as claimed in claim 1, wherein the transformer icon can transform more than one icon at a time.

3. An apparatus as claimed in claim 1 or claim 2, including data structure means, associated with the transformer icon, for storing rules defining the characteristics of the transformation operation.

4. An apparatus as claimed in any preceding claim, including means for transforming first information associated with the input icon into second information associated with an output icon.

5. An apparatus as claimed in any preceding claim, including means for transforming an input icon into a plurality of output icons.

6. An apparatus as claimed in any preceding claim, including means for transforming a plurality of input icons into a single output icon.

7. An apparatus as claimed in any preceding claim, including means for transforming a plurality of input icons into a plurality of output icons.

Patentansprüche

1. Rechnergerät, enthaltend eine Anzeigevorrichtung (38), Informationsspeichermittel (14, 16, 20) und einen Prozessor (10), der auf eine vorbestimmte Anwendereingabeoperation anspricht, um gespeicherte Informationen gemäß der Position zugeordneter Ikone auf dem Bildschirm der Anzeigevorrichtung (38) zu verarbeiten, dadurch gekennzeichnet, daß wenigstens eines dieser Ikone ein Umwandlungs-Ikon ist, das so angeordnet ist, daß es eine Umwandlungsoperation anzeigt, die an gespeicherten, einem oder mehreren Eingabe-Ikonen (200) zugeordneten Informationen durchgeführt werden kann, und es ermöglicht, daß die Umwandlungsoperation durch eine vorbestimmte Anwendereingabeoperation aufgerufen wird, wenn das Eingabe-Ikon bzw. jedes Eingabe-Ikon über das Umwandlungs-Ikon (210) positioniert wird, wobei das Gerät Mittel aufweist, um ein oder mehrere Ausgabe-Ikonen (230) zu generieren, die sich vom Umwandlungs-Ikon unterscheiden, das der umgewandelten Information zugeordnet ist.

2. Ein Gerät gemäß Anspruch 1, in dem das Umwandlungs-Ikon mehr als ein Ikon gleichzeitig umwandeln kann.

3. Ein Gerät gemäß Anspruch 1 oder Anspruch 2, einschließlich Datenstrukturmittel, die dem Umwandlungs-Ikon zugeordnet sind, um Regeln abzuspeichern, die die Merkmale der Umwandlungsoperation definieren.

4. Ein Gerät gemäß einem beliebigen der vorstehenden Ansprüche, einschließlich Mittel zum Umwandeln erster, dem Eingabe-Ikon zugeordneter Informationen, in zweite, einem Ausgabe-Ikon zugeordnete Informationen.

5. Ein Gerät gemäß einem beliebigen der vorstehenden Ansprüche, einschließlich Mittel zum Umwandeln eines Eingabe-Ikons in eine Vielzahl von Ausgabe-Ikone.

6. Ein Gerät gemäß einem beliebigen der vorstehenden Ansprüche, einschließlich Mittel zum Umwandeln einer Vielzahl von Eingabe-Ikonen in ein einziges Ausgabe-Ikon.

7. Ein Gerät gemäß einem beliebigen der vorstehenden Ansprüche, einschließlich Mittel zum Umwandeln einer Vielzahl von Eingabe-Ikonen in eine Vielzahl von Ausgabe-Ikonen.

Revendications

1. Appareil informatique comprenant un dispositif d'affichage (38), un moyen de mémorisation d'informations (14, 16, 20) et un processeur (10) répondant à une opération d'entrée prédéterminée par l'utilisateur pour traiter des informations mémorisées conformément à l'emplacement d'icônes associées sur l'écran du dispositif d'affichage (38), caractérisé en ce qu'au moins une desdites icônes est une icône de transformation conçue pour représenter une opération de transformation qui peut être effectuée sur les informations mémorisées associées à une ou plusieurs icônes d'entrée (200) et pour permettre à l'opération de transformation d'être appelée par une opération d'entrée prédéterminée par l'utilisateur lorsque la ou chaque icône d'entrée est positionnée sur l'icône de transformation (210), l'appareil comprenant un moyen pour générer une ou plusieurs icônes de sortie (230) distincte (s) de l'icône de transformation et associée(s) aux informations transformées.

2. Appareil selon la revendication 1, dans lequel l'icône de transformation peut transformer plus qu'une icône en une fois.

3. Appareil selon la revendication 1 ou la revendication 2, incluant un moyen de structure de données associé à l'icône de transformation pour mémoriser des règles définissant les caractéristiques de l'opération de transformation.

4. Appareil selon l'une quelconque des revendications précédentes, comprenant un moyen pour transformer des premières informations associées à l'icône d'entrée en secondes informations associées à une icône de sortie.

5. Appareil selon l'une quelconque des revendications précédentes, comprenant un moyen pour transformer une icône d'entrée en une pluralité d'icônes de sortie.

6. Appareil selon l'une quelconque des revendications précédentes, comprenant un moyen pour transformer une pluralité d'icônes d'entrée en une seule icône de sortie.

7. Appareil selon l'une quelconque des revendications précédentes, comprenant un moyen pour transformer une pluralité d'icônes d'entrée en une pluralité d'icônes de sortie.

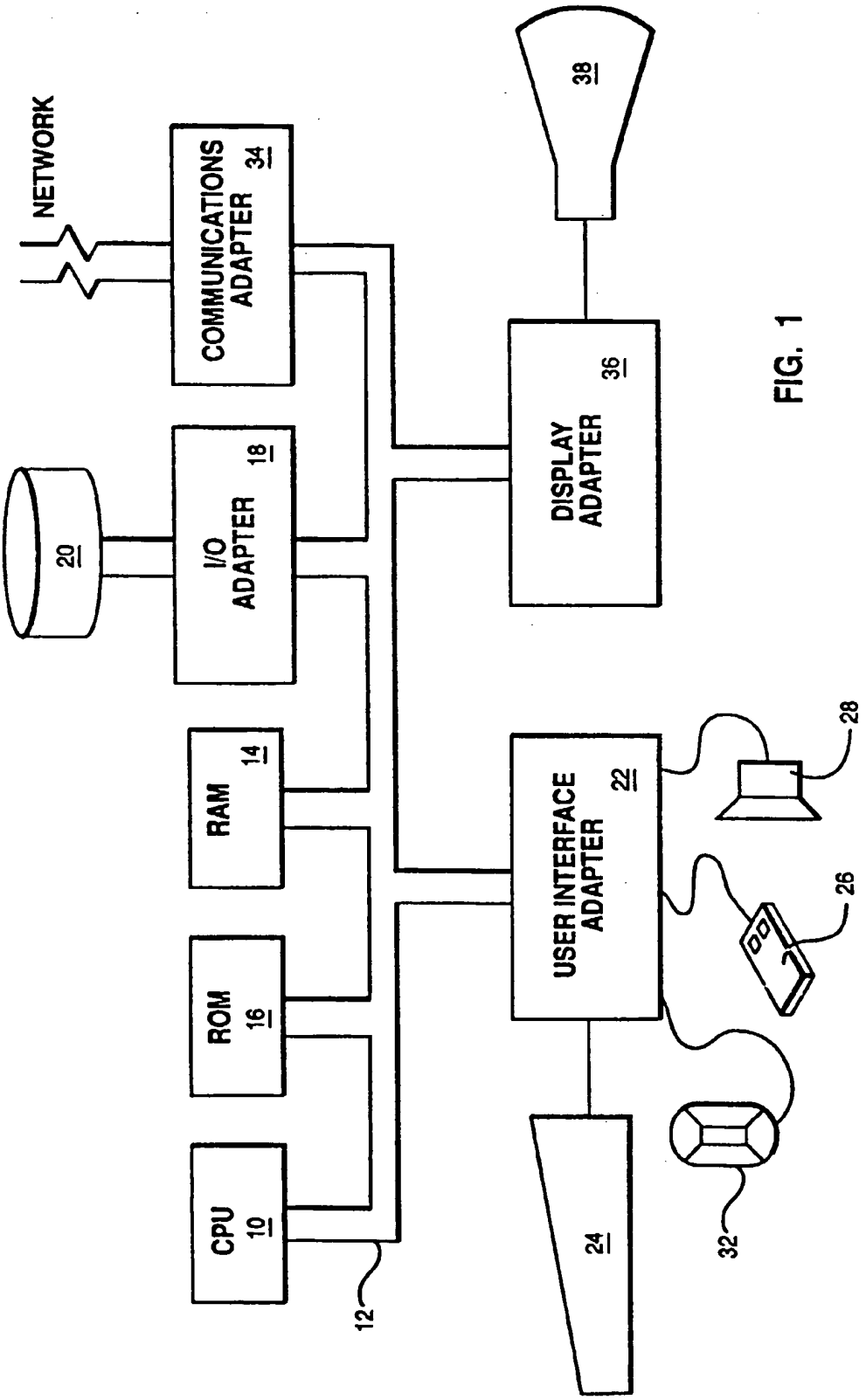
NETWORK
COMMUNICATIONS ADAPTER 34
38
DISPLAY ADAPTER 36
I/O ADAPTER 18
20
RAM 14
ROM 16
CPU 10
12
USER INTERFACE ADAPTER 22
28
26
24
32

FIG. 1

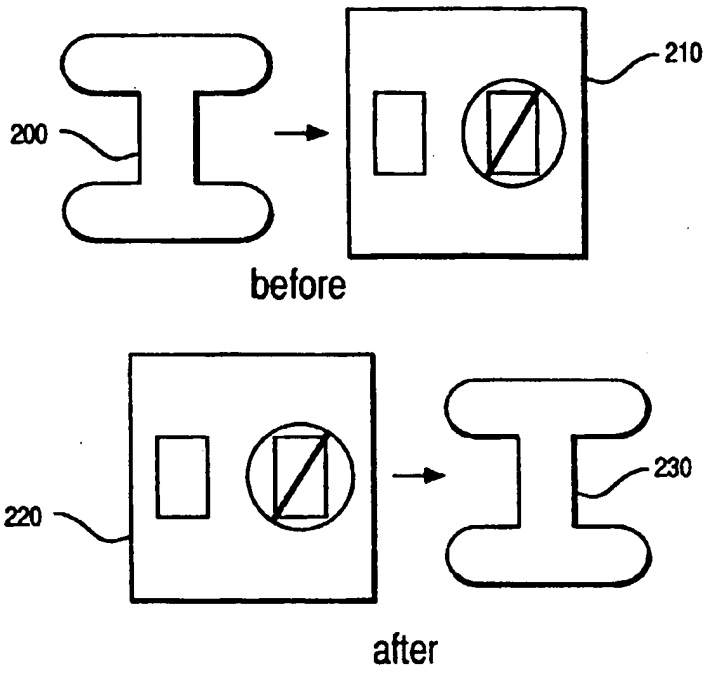

FIG. 2

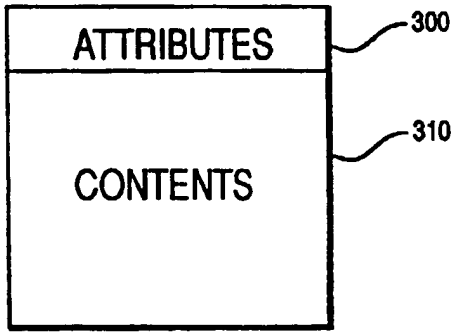

FIG. 3

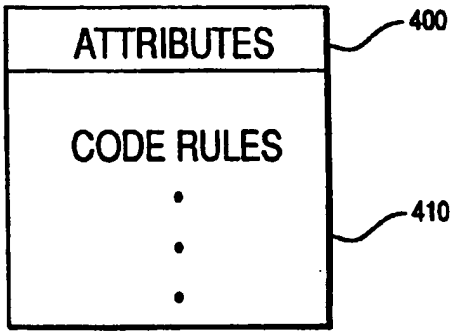

FIG. 4

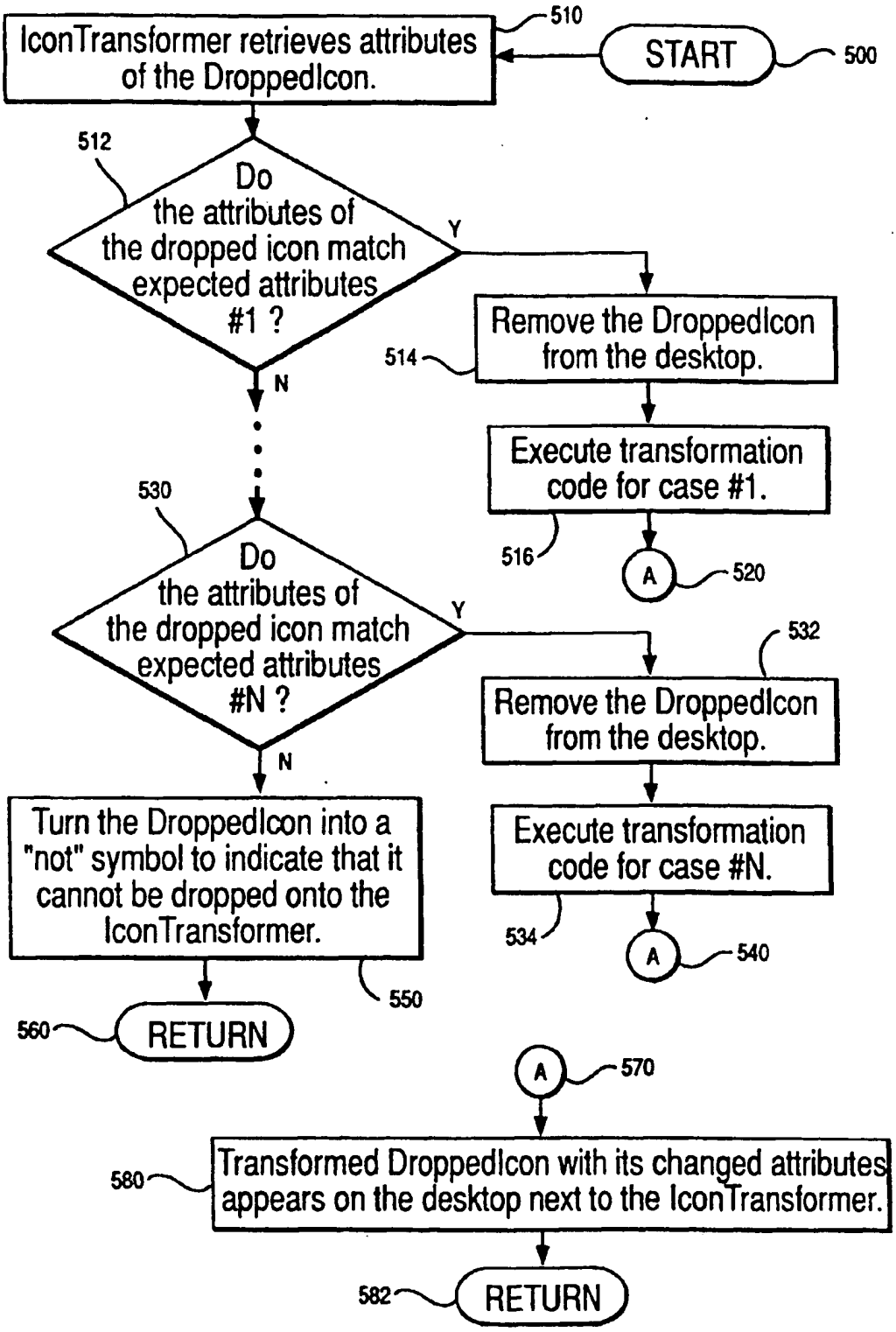
START
500
IconTransformer retrieves attributes of the DroppedIcon.
510
Do the attributes of the dropped icon match expected attributes #1 ?
512
Y
N
Remove the DroppedIcon from the desktop.
514
Execute transformation code for case #1.
516
A
520
Do the attributes of the dropped icon match expected attributes #N ?
530
Y
N
Remove the DroppedIcon from the desktop.
532
Execute transformation code for case #N.
534
A
540
Turn the DroppedIcon into a "not" symbol to indicate that it cannot be dropped onto the IconTransformer.
550
RETURN
560
A
570
Transformed DroppedIcon with its changed attributes appears on the desktop next to the IconTransformer.
580
RETURN
582

FIG. 5

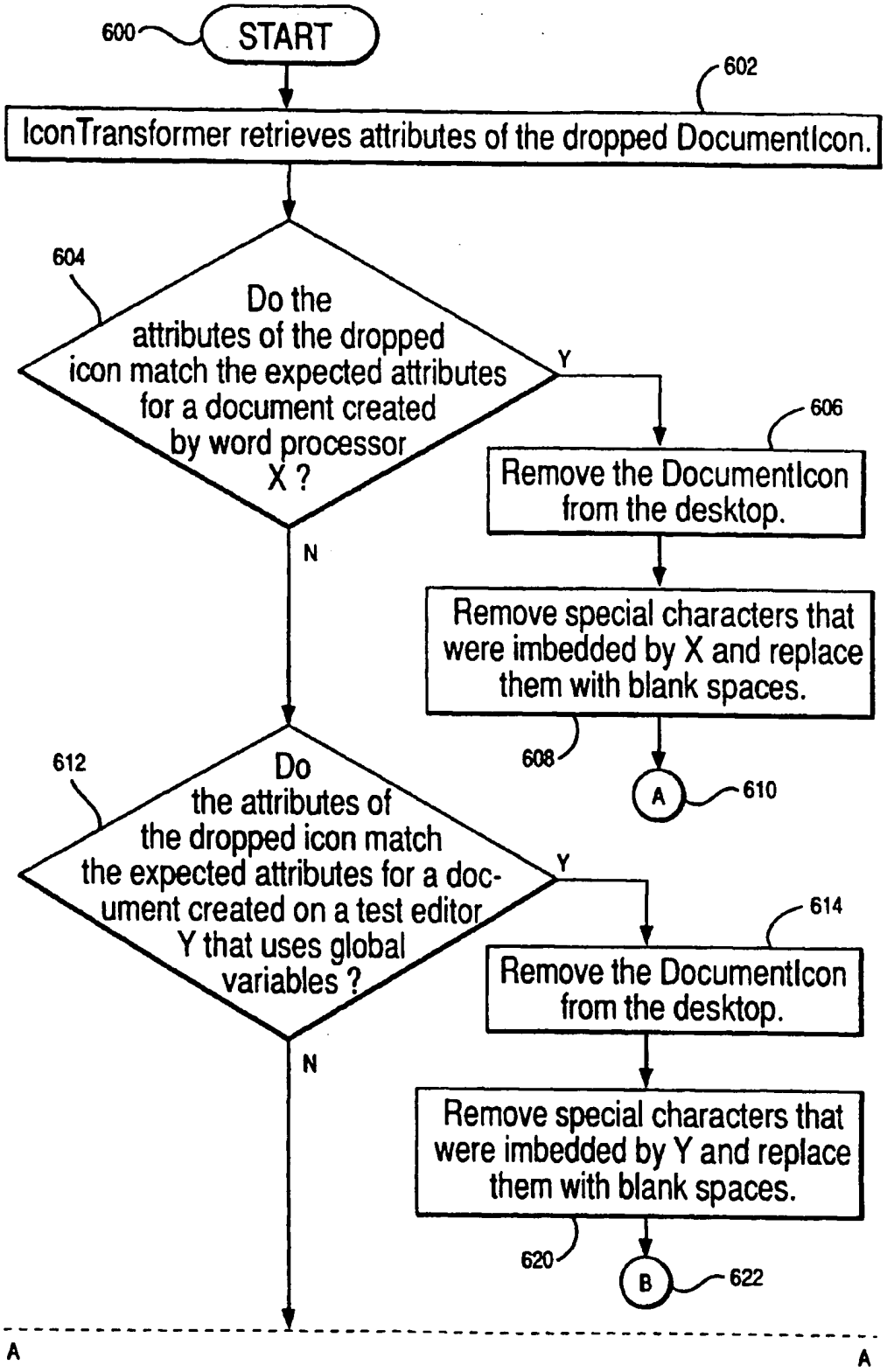
600
START
602
IconTransformer retrieves attributes of the dropped DocumentIcon.
604
Do the attributes of the dropped icon match the expected attributes for a document created by word processor X ?
Y
606
Remove the DocumentIcon from the desktop.
N
Remove special characters that were imbedded by X and replace them with blank spaces.
608
A
610
612
Do the attributes of the dropped icon match the expected attributes for a document created on a test editor Y that uses global variables ?
Y
614
Remove the DocumentIcon from the desktop.
N
Remove special characters that were imbedded by Y and replace them with blank spaces.
620
B
622
A
A

FIG. 6A

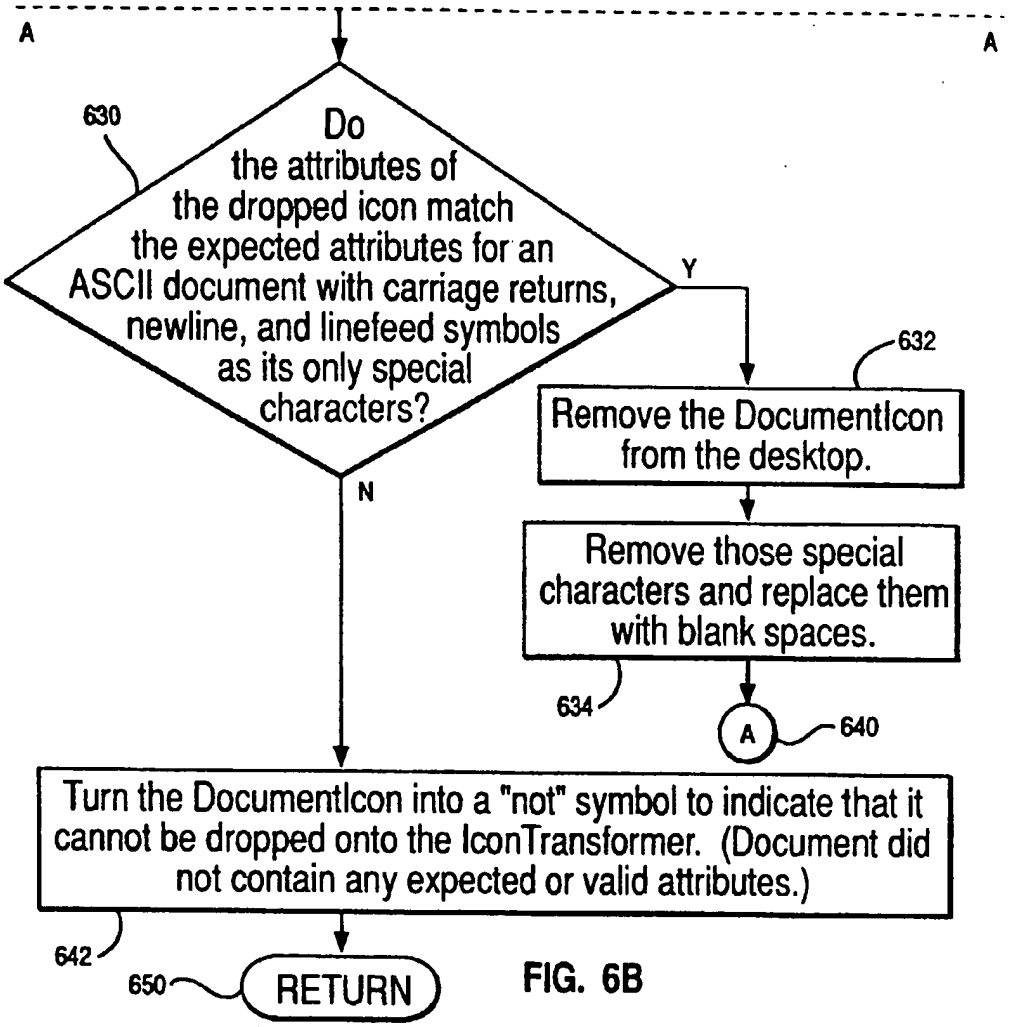
A
A
630
Do the attributes of the dropped icon match the expected attributes for an ASCII document with carriage returns, newline, and linefeed symbols as its only special characters?
Y
N
632
Remove the DocumentIcon from the desktop.
Remove those special characters and replace them with blank spaces.
634
A
640
Turn the DocumentIcon into a "not" symbol to indicate that it cannot be dropped onto the IconTransformer. (Document did not contain any expected or valid attributes.)
642
650
RETURN

FIG. 6B

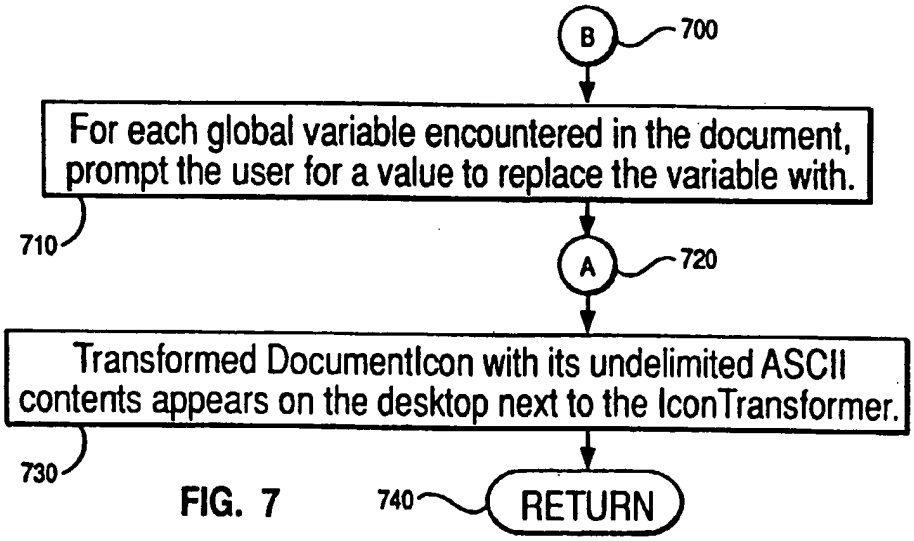
B
700
For each global variable encountered in the document, prompt the user for a value to replace the variable with.
710
A
720
Transformed DocumentIcon with its undelimited ASCII contents appears on the desktop next to the IconTransformer.
730
740
RETURN

FIG. 7